CILISSA

Release 0.7.1

Kamil Marut

Feb 25, 2022

CONTENTS:

CILISSA stands for **C**omputer **I**mage **Li**keness A**ss**essing **A**utomation.

CHAPTER

ONE

OVERVIEW

CILISSA allows for the use of various metrics to perform full-reference image comparisons.

It features the most popular full-reference image quality metrics, image transformations and translations. CILISSA is also very extensible and new operations can be easily added.

CILISSA has an optional Qt-based graphical interface that lets you experiment with various operations, their orders and properties.

1.1 Installation

1.1.1 Python Support

CILISSA supports these Python versions.

Python	3.10	3.9	3.8	3.7	3.6
CILISSA < 1.0	Yes	Yes	Yes	Yes	

1.1.2 Basic Installation

1.2 Examples

1.2.1 Image pair analysis

The following script compares 2 images using the SSIM metric via its `analyze` method or uses both the MSE and SSIM metrics **in order** using the `OperationsList` class.

```
from cilissa.images import Image, ImagePair
from cilissa.metrics import SSIM, MSE
from cilissa.operations import OperationsList

image1 = Image("path/to/original/image")
image2 = Image("path/to/other/image")
image_pair = ImagePair(image1, image2)

# Compare using standalone metric
ssim = SSIM(channels_num=3)
result = ssim.analyze(image_pair)
```

(continues on next page)

(continued from previous page)

```
# Or use OperationsList
mse = MSE()
operations = OperationsList([mse, ssim])
results = operations.run_all(image_pair)
```

1.2.2 Image transformation

The following script transforms a single image with the `Blur` transformation using its `transform` method. Transformations can be chained and mixed in the `OperationsList` class in the same way as metrics.

```
from cilissa.images import Image
from cilissa.transformations import Blur

image = Image("path/to/original/image")

# Transform using standalone transformation
blur = Blur(gaussian=False, sigma=2.0)
result = blur.transform(image)
```

1.3 Reference

1.3.1 `cilissa.images` - Image structures

Classes

class cilissa.images.**Image**(*image: Union[pathlib.Path, str, numpy.ndarray]*, *name: Optional[str] = None*)

np.ndarray wrapper, a core structure in CILISSA

as_float(*self*)

Converts the image to `np.ndarray` of floats

as_int(*self*)

Converts the image to `np.ndarray` of ints

copy(*self*)

Copies and returns the image

from_array(*self*, *image_array: numpy.ndarray*, *at: Optional[Tuple[slice, slice]] = None*)

Replaces the underlying image array with given *np.ndarray*

load(*self*, *image_path: Union[pathlib.Path, str]*)

Loads the image from given path

Uses cv2.imdecode instead of cv2.imread to handle unicode characters in path

Parameters **image_path** (*Path/str*) – Path where the image is located.

save(*self*, *save_path: Union[pathlib.Path, str] = ''*)

Saves the image

Parameters **save_path** (*Path/str*) – Path to save the image at. Must contain the filename with extension. If empty string, then will save to the path the image was loaded from (if available)

show(*self*)
Opens a CV2 window and displays the loaded image.

Exits when user presses ESCAPE or closes window manually.

class cilissa.images.**ImagePair**(*im1: Image*, *im2: Image*, *roi: Optional[*cilissa.roi.ROI*] = None*, *use_roi: bool = True*)
A pair of 2 *cilissa.images.Image*. Analysis is performed using this class.

If any of the attributes in the image pair are mismatched, the attribute of the reference image will be used if necessary.

as_floats(*self*)
Returns a tuple with both images as `np.ndarray` of floats

im1
Reference image against which quality is measured

im2
Image whose quality is to be measured

swap(*self*)
Swaps the reference and input images in place.

class cilissa.images.**ImageCollection**(*items: List[Any] = []*)
A collection of one or more `cillisa.images.ImagePair`.

Operations performed on `cillisa.images.ImagePair` can be applied to the whole collection.

1.3.2 `cilissa.metrics` - Image quality metrics

Classes

class cilissa.metrics.**MSE**(*rmse: bool = False*)
Mean squared error (MSE)

Average squared difference between the estimated values and the actual value.

References

- https://en.wikipedia.org/wiki/Mean_squared_error

class cilissa.metrics.**PSNR**(*max_pixel_value: Optional[int] = None*)
Peak signal-to-noise ratio (PSNR)

Ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

References

- https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

class cilissa.metrics.**SSIM**(*channels_num: Optional[int] = None*, *sigma: float = 1.5*, *truncate: float = 3.5*, *K1: float = 0.01*, *K2: float = 0.03*)

Structural similarity index measure (SSIM)

The SSIM Index quality assessment index is based on the computation of three terms, namely the luminance term, the contrast term and the structural term. The overall index is a multiplicative combination of the three terms.

Parameters **channels_num** (*int/None*) – If None, image is assumed to be grayscale (single channel). Otherwise the number of channels should be specified here.

Returns Overall quality measure of the entire image (MSSIM)

Return type float

References

- https://en.wikipedia.org/wiki/Structural_similarity
- https://ece.uwaterloo.ca/~z70wang/publications/ssim.pdf

class cilissa.metrics.**UIQI**(*channels_num: Optional[int] = None*, *block_size: int = 8*)

Universal Image Quality Index (UIQI)

Combines loss of correlation, luminance distortion and contrast distortion. Predecessor of SSIM metric.

References

- https://ece.uwaterloo.ca/~z70wang/publications/quality_2c.pdf

class cilissa.metrics.**VIFP**(*channels_num: Optional[int] = None*, *sigma: float = 2.0*)

Pixel Based Visual Information Fidelity (VIF-P)

Employs natural scene statistical (NSS) models in conjunction with a distortion (channel) model to quantify the information shared between the test and the reference images.

The pixel domain algorithm is not described in the paper below. This is a computationally simpler derivative of the algorithm presented in the paper, based on the original Matlab implementation.

References

- http://live.ece.utexas.edu/publications/2004/hrs_ieeetip_2004_imginfo.pdf
- https://github.com/utlive/vif_pixel

class cilissa.metrics.**SAM**(*to_degrees: bool = True*)

Spectral Angle Mapper (SAM)

Physically-based spectral classification that uses an n-D angle to match pixels to reference spectra.

This technique is relatively insensitive to illumination and albedo effects.

References

- https://ntrs.nasa.gov/citations/19940012238

1.3.3 `cilissa.transformations` - Image transformations and distortions

Classes

class cilissa.transformations.**Blur**(*gaussian: bool = True*, *kernel_size: Tuple[int, int] = (0, 0)*, *sigma: float = 1.0*)

Blurs an image.

Parameters

- **gaussian** (*bool*) – If True, uses a Gaussian filter to blur the image. If False, uses normalized box filter.
- **kernel_size** (*Tuple[int, int]*) – Gaussian/blurring kernel size. Elements in tuple can differ but they both must be positive and odd. If using Gaussian filter they can be zero's and then they are computed from sigma.
- **sigma** (*float*) – Gaussian kernel standard deviation in X direction. Used only with Gaussian filter.

References

- https://docs.opencv.org/4.5.2/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37
- https://docs.opencv.org/4.5.2/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

class cilissa.transformations.**Sharpen**(*amount: float = 1.5*, *kernel_size: Tuple[int, int] = (0, 0)*, *sigma: float = 1.0*)

Sharpens an image using an unsharp mask.

Parameters

- **amount** (*float*) – Amount of sharpening applied.
- **threshold** (*int*) – Threshold for the low-constrast mask. Pixels for which the difference between the input and blurred images is less than threshold will remain unchanged.
- **kwargs** (*Any*) – Arguments passed as kwargs will be passed to the blur transformation.

References

- https://en.wikipedia.org/wiki/Unsharp_masking

class cilissa.transformations.**Linear**(*contrast: Optional[Union[float, int]] = None*, *brightness: Optional[Union[float, int]] = None*)

Changes brightness of the image with a simple linear transformation.

g(x) = a * f(x) + b, where *a* controls contrast and *b* controls brightness

Brightness refers to the overall lightness or darkness of the image. Increasing the brightness every pixel in the frame gets lighter.

Contrast is the difference in brightness between objects in the image. Increasing the contrast makes light areas lighter and dark area in the frame becomes much darker.

Parameters

- **contrast** (*int/float/None*) – Value by which to change the contrast. 1 and None is the original image. A float from interval (0, 1) reduces the contrast. Values above 1 increase the contrast.
- **brightness** (*int/float/None*) – Value by which to change the brightness. 0 and None is the original image. Negative values reduce the brightness. Positive values increase the brightness.

References

- https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html

class cilissa.transformations.**Translation**(*x: int = 0*, *y: int = 0*)

Shifts an image by given amount in pixels along X and/or Y axis.

Uses an affine transformation to perform image translation.

Parameters

- **x** (*int*) – Value (in px) to move the image along X-axis. Positive - right, negative - left.
- **y** (*int*) – Value (in px) to move the image along Y-axis. Positive - down, negative - up.

References

- https://en.wikipedia.org/wiki/Affine_transformation

class cilissa.transformations.**Equalization**(*nbins: int = 256*)

Constrast adjustment using histogram equalization.

CV2 equalization is limited to 8-bit images so we use the NumPy CDF function.

References

- https://en.wikipedia.org/wiki/Histogram_equalization
- https://web.archive.org/web/20151219221513/http://www.janeriksolem.net/2009/06/histogram-equalization-with-python-and.html # noqa

1.3.4 cilissa.operations - Base operation classes

Classes

class cilissa.operations.**ImageOperation**

Base class for all operations that can be performed on an image.

Display name and name used in various dicts is deduced from the class name.

class cilissa.operations.**Metric**

Base class for creating new metrics to use in the program.

class cilissa.operations.**Transformation**

Base class for creating new transformations to use in the program.

class cilissa.operations.**OperationsList**(*items: List[Any] = []*)
Simple list that can be reordered.

Behaviour similar to *list*, but does not derive from it to avoid lay-out conflict with QObject

Implements *get_order* and *change_order* methods to manage the list's order.

1.3.5 cilissa.results - Operations results

Classes

class cilissa.results.**Result**
Helper class that provides a standard way to create an ABC using inheritance.

class cilissa.results.**ResultGenerator**(*results: List[Result]*)

1.3.6 cilissa.roi - Region of interest

Classes

class cilissa.roi.**ROI**

1.3.7 cilissa.parsers - Parse different formats to CILISSA objects

Functions

cilissa.parsers.**parse_operations_from_str**(*operations: List[str]*, *kwargs: List[Any]*)
Parses operations and their parameters from string input.

Parameters use the following format:

<operation-name>-<parameter-name>=<value>

where *parameter-name* uses hyphens (-) instead of underscores (_).

cilissa.parsers.**parse_operations_from_json**(*fp: TextIO*)
Parses operations and their parameters from a JSON file.

Expected dictionary structure: *[{"name": "ssim", "parameters": {"channels_num": 3, "sigma": 1.5}}]*

cilissa.parsers.**parse_roi**(*string: str*)
Parses ROI from string.

Expected string format: *0x0,512x512* (width x height)

1.3.8 cilissa.helpers - Helper functions

Functions

cilissa.helpers.**get_max_pixel_value**(*dtype: numpy.dtype*)

cilissa.helpers.**apply_to_channels**(*im1: numpy.ndarray*, *im2: numpy.ndarray*, *func: Callable*, *channels_num: int*)

cilissa.helpers.**crop_array**(*array: numpy.ndarray*, *crop_width: int*)
Crop input *array* by *crop_width* in every dimension

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX